Aula 12

12 – Instalando software

Como sabemos, o GNU/Linux trabalha em quase sua totalidade com softwares e aplicativos em código livre. Isso leva a concluir que deve ser de fácil acesso e conhecimento a instalação e configuração da maioria das aplicações disponíveis. Em geral, basta se obter o código fonte da aplicação em interesse e compilá-la no sistema local.

Isso é verdade, mas em termos de praticidade e administração do sistema torna-se um processo complexo e tedioso fazer o controle de todo o software instalado através de compilação direta de um código fonte.

Para facilitar o controle e melhorar o processo de instalação de aplicativos em sistemas Linux; existem alguns softwares e ferramentas que auxiliam ao administrador de sistemas. O objetivo desta aula é analisá-los e compreender o mecanismo elementar de seu funcionamento.

Pensando de forma simples e objetiva, para se instalar um software qualquer, é necessário compilarmos seu código e depois copiarmos os arquivos executáveis e objetos resultantes nos locais corretos. O destino em geral será algum diretório de binários (/bin) ou de módulos (/lib).

Agora, imagine que já você já dispõe de um programa executável ou objeto pronto e deseje somente instalá-lo. Para isso, bastaria copiar os devidos arquivos nos destinos esperados. Nesse ponto é que entram as ferramentas de auxílio. No caso de softwares já compilados, costuma-se chamar de pacotes, um conjunto de arquivos agrupados, prontos a serem utilizados. São esses conjuntos de programas, bibliotecas ou documentações que devem copiadas aos diretórios destinos.

O *Red Hat* surgiu com uma ferramenta chamada *rpm* (*Red Hat Package Manager*), já o Debian tem como ferramenta base, o *dpkg* (*Debian Package*). Ambas ferramentas possuem uma base de dados que controla a versão e estado dos softwares instalados em seu sistema. Obviamente esse controle só é realizado mediante àqueles pacotes ou softwares instalados através dessas ferramentas.

Ainda para deixar mais prático e eficiente a instalação desses softwares, foram desenvolvidos métodos para atualizar e instalar os pacotes em rede. A grande qualidade dessas ferramentas talvez se concentre na facilidade de se poder consultar e obter os softwares livres na rede. Os mais conhecidos e utilizados são o *APT* (Advanced Package Tool) e *YaST* (*Yet another Setup Tool*).

A seguir são passados os principais comandos para utilização dessas ferramentas, bem como a compilação de código fonte.

12.1 – A partir do código fontes

Fazer a instalação de softwares relativamente pequenos é simples e talvez não requeira tanta experiência para um usuário familiarizado com os interpretadores de comandos:

#> gcc -o arquivo_objeto arquivo_fonte.c

Este comando resultaria na compilação através do *Gnu C compiler* para o código fonte de *arquivo_fonte.c*. O executável resultante seria *arquivo_objeto*.

Até este ponto, parece ser simples. O problema começa aumentar quando se tem um software complexo onde se tem diversas dependências desde módulos e bibliotecas .

Pensando em minimizar o que seria um problema até para usuários avançados, é que se desenvolveu a ferramenta *make*, que como o próprio nome já diz: *Faz* ou *constrói* um software.

O *make*, também é bastante simples, funcionando com um arquivo de configuração denominado de *Makefile*. A aplicação *make* lê este arquivo como parâmetro de entrada e executa conforme uma sequência de regras bem estabelecidas o passo a passo da compilação de um software. O *Makefile (*escrito com inicial maiúscula - Makefile) fica no diretório corrente onde a aplicação *make* é invocada e contém uma série de regras de compilação, tendo no mínimo um binário alvo e os códigos dependentes.

```
#> ./configure
#> make
#> make test
#> make install
```

Nos passos acima, a primeira execução é aquela onde é feita uma configuração, que nada mais é do que a verificação de dependências como bibliotecas por exemplo. Além dessa verificação, o passo de configuração serve para criar o *Makefile* e determinar ao mesmo quais as variáveis a serem consideradas na compilação e instalação. Exemplos dessas variáveis seriam o diretório residente da aplicação, a utilização de determinadas bibliotecas, suporte ou ativação de alguns módulos e todas variáveis que um determinado software possa permitir. O comando *configure* serve basicamente, para construir o *Makefile* e atribuir os parâmetros possíveis para as variáveis permitidas no momento da compilação. Em geral, todos desenvolvedores fornecem *scripts* de configure, pois com eles, seus softwares se tornam mais flexíveis, podendo ter seus parâmetros alterados antes da compilação.

No segundo passo, o comando *make* constrói o sistema baseado no arquivo de *Makefile,* que por definição reside no mesmo diretório donde se está executando a aplicação *make*. Nesse ponto, as informações são lidas do *Makefile* e os comandos de compilação são invocados levando em consideração essas orientações.

Os dois últimos passos são intuitivos, pois o penúltimo faz um teste. Ou seja, antes de executar qualquer cópia, faz uma simulação da cópia dos arquivos para verificar se algo de errado ocorreria ou ainda pode servir para indicar onde os binários e arquivos finais seriam colocados. São poucos distribuidores que adicionam a regra de *test* em seus *Makefiles*, mas ainda assim, quando possível, é recomendável utilizá-la.

Por fim, vem o passo de instalação (*make install*), que serve para concluir o processo, copiando os arquivos para seus respectivos destinos. Isto é, os manuais, supostamente para */usr/share/man*, os binários para */usr/local/bin*, os arquivos de configuração para /*etc/* e assim por diante. Em geral, esse passo é executado com privilégios de administrador para evitar problemas de permissão.

12.1.2 – Construindo um *makefile* simples

Como foi citado anteriormente, o *make* é uma ferramenta que auxilia na construção de softwares complexos, que contenham diversas dependências e códigos relacionados. Para isso o *Makefile* possui regras e variáveis. As duas principais sintaxes necessárias para se construir um *Makefile* são: A regra para um objeto alvo, variáveis e regras para ações específicas.

Para dependências e objetos, a sintaxe deve seguir a seguinte regra de construção:

```
alvo : dependencias1.o dependencia2.o dependencia3.o . . .
        comandos
```

Onde, o alvo é o binário ou objeto que ser deseja obter mediante a construção dos vários objetos relativos. Em geral, o objeto que contiver a classe principal também vai ter o mesmo nome do objeto alvo. Ainda citando os objetos, é bom perceber que todos eles estão já no formato de objeto (.o), mas deve-se entender que estes objetos não existem até que sejam compilados por ordem do *make,* criados a partir dos códigos fontes C ou C++.

Logo abaixo, segue a linha de comandos a serem executados após a criação do alvo. Esta linha deve ser endentada por uma tabulação e não por um espaço simples.

A outra regra para construção de *Makefile* é a definição de variáveis. Esta deve seguir uma convenção de variáveis padrões como por exemplo opções de compilação, compilador escolhido entre outras.

A sintaxe é a mesma como de um programa ou algoritmo:

```
Variável = valor
```

Conhecendo somente esses dois elementos formadores de um *Makefile,* é possível construir algo bem simples. Imagine um programa bastante simples, sem utilidade prática, apenas para prova de conceito. Este programa vai calcular a multiplicação de dois números dados pelo usuário.

Os códigos seriam:

```
# conta.c
float conta (float x, float y)
{
        return x*y;
}
```

```
#atributos.c
#include <stdio.h>
int main()
{
        float conta(float a, float b);
        float a,b,c;

        printf("Forneca uma valor para o primeiro fator\n");
        scanf("%f",&a);
        printf("Forneca uma valor para o segundo fator\n");
        scanf("%f",&b);
        c = conta(a,b);
        printf("O resultado é %f \n",c);
        return 0;
}
```

Seria possível construir um *Makefile* para o programa *conta* a partir dos códigos de seus objetos *conta.c* e *atributos.c*. O arquivo *Makefile* poderia ser assim escrito:

```
CC=gcc
CFLAGS=-Wall

conta : conta.o atributos.o

clean:
    rm -f conta conta.o atributos.o

install:
    cp conta /usr/local/bin/
```

O arquivo poderia ser lido como o arquivo de construção do programa conta. Onde o alvo conta possui as dependências *conta.o* e *atributos.o* (retirados de *conta.c* e *atributos.c*) e não possui nenhum comando.

Já os alvos *clean* e *install* não possuem dependências, porém somente comandos. O comando a ser executado para o alvo *clean* é a remoção de todos objetos e para o alvo *install* é a copia do binário conta resultante para o diretório */usr/bin*.

As variáveis, não menos importantes, são a *CC* que define qual compilador a ser utilizado para a geração dos binários e objetos e a variável *CFLGAS* que define quais opções utilizadas para a compilação. O valor *Wall* significa que ao compilar os programas, o *gcc* deverá exibir na saída padrão qualquer alerta (*warning*) durante esse processo. Esta opção é importante e sempre é bom utilizá-la para evitar problemas de código.

Com isso, é possível criar um *Makefile* com os quesitos mínimos. Obviamente que, para um código tão simples quanto o utilizado, não há motivos em se usar *Makefile*. Entretanto essa é uma ótima ferramenta quando se necessita gerenciar e distribuir um software complexo que requeira muitas variáveis de compilação e diversas opções de objetos interligados.

12.2 – Pacotes binários

Como já falado na introdução, as ferramentas *dpkg* e *rpm* são as mais usadas para aqueles sistemas baseados em *Debian* e *Red Hat* respectivamente. Para cada um deles, veremos principais comandos ou arquivos relevantes.

12.2.1 - RPM

O *RPM* controla as aplicações instaladas em um sistema do tipo *Red Hat* ou suas variantes de forma bastante prática e simples. Ele é constituído de uma base de dados que indica quais são as aplicações instaladas em um sistema e suas informações.

Dentre outras informações a base de dados armazena datas de aplicações instaladas, versões, localizações dos arquivos, etc. É bastante utilizada para instalar pacotes binários prontos, mesmo assim permite a instalação de códigos em rpm.

Todos os arquivos em formatos de pacotes são distinguidos pela extensão rpm e os principais comandos de manipulação destes arquivos são:

rpm –i (instala)
rpm –u (upgrade)
rpm –e (remove)
rpm –qa (exibe pacotes instalados)
rpm –qi (informações sobre um pacote instalado)
rpm –ql (lista os arquivos de um pacote instalado)
rpm –qf (indica a qual pacote instalado é proveniente um arquivo)
rpm –qpi (exibe informações de um pacote não instalado)
rpm –qpl (exibe arquivos que fazem parte de um pacote não instalado)

Os repositórios mais importantes de *RPM* são o HTTP://www.rpmfind.net ou HTTP://rpm.pbone.net

12.2.2 - DPKG

O *dpkg* também se assemelha muito ao *rpm* pelo fato de utilizar uma base de dados e possuir comandos basicamente de consulta e instalação. As principais opções e arquivos são:

#> dpkg -i <pacote.deb>
Instala um pacote

#> dpkg –configure <pacote>
Configura um pacote instalado

#> dpkg -r <pacote>
Remove um pacote

#> dpkg –purge <pacote>
Remove um pacote incluindo arquivos de configuração

#> dpkg -r <pacote>
Remove um pacote

#> dkpg -l <pacote>
Lista estado de um pacote

#> dpkg -c <pacote.deb>
Mostra conteúdo de um arquivo de pacote

#>dpkg -L <pacote>
Lista arquivos instalados a partir de um pacote

#dpkg -I <pacote.deb>
Exibe informações sobre um arquivo de pacote

#>dpkg -s <pacote>
Exibe status sobre um pacote específico

/etc/dpkg/dpkg.cfg
Arquivo de configuração para o dpkg. Opções padrão para a execução da aplicação dpkg.

/var/log/dpkg.log
Log de execução do dpkg

/var/lib/dpkg/status
Estado e informação de todos pacotes instalados, selecionados ou removidos.

/var/lib/dpkg/avaiable
Informação dos pacotes já obtidos e disponíveis.

/var/lib/dpkg/info/

O subdiretório info contém os pacotes instalados e as informações sobre os arquivos que cada um deles instalou no sistema.

12.2.3 – Pacotes online: APT

O *APT* (*Advance Package Tool)* é apenas uma ferramenta auxiliar que executa sobre o *dpkg*. A idéia essencial do APT é manter uma fonte de repositório de software que pode ser acessada para instalar ou trazer informações dos softwares disponíveis.

Basicamente possui os mesmo comandos que o *dpkg*, mas com o acréscimo das configuração dos repositórios. São estes os principais comandos e arquivos:

- ***/etc/apt/souces.list***

Este arquivo é quem define as fontes para as quais a aplicação *apt* vai buscar os índices dos softwares disponíveis e também baixar os pacotes.

Exemplos de linhas genéricas do arquivo *sources.list* podem ser expressas como:

deb http://site.http.org/debian distribuição área/seção1 seção2 seção3
deb-src http://site.http.org/debian distribuição área/seção1 seção2 seção3

O parâmetro inicial *deb* indica que o repositório é uma origem de arquivos compilados com extensão .deb e o *deb-src* indica uma origem de arquivos de código fonte, cuja extensão é dsc. Logo em seguida, temos as identificações de localizações ou *URI* (*Universal Resource Identifier*) que podem ser desde discos rígidos até arquivos em um servidor *ftp* ou *http*. Mais especificamente, as possibilidades para uma *URI* podem ser: *file*, *cdrom*, *http*, *ftp*, *copy* e *rsh*/*ssh*.

O terceiro parâmetro informa a distribuição e seu grau de confiabilidade. Como já havia sido comentado, nas primeiras aulas, para o Debian, existem nomes de personagens do filme *Toy Story* para cada uma das distribuições e para o grau de confiabilidade existem os nomes *testing*, *unstable* e *stable*.

A versão *testing* é aquela que ainda não está madura o suficiente, mas está na fila, candidatando-se a uma versão estável, ou seja *stable*. Por outro lado, a versão *unstable* determina aqueles pacotes em desenvolvimento, com alterações e testes contínuos. Todas versões *unstable* alternativamente são referenciadas com o nome do personagem s*id*.

Nos itens de área, que oficialmente são 3, podemos encontrar as seguintes classificações de tipos de arquivos segundo seu padrão de licenciamento. Sendo elas

- *main*: principal área de arquivo e deveria ser o padrão Debian, comportando todo software que é gratuiro e segue as recomendações da DFSG (Debian Free Software Guidelines)
- *contrib*: utiliza softwares ou módulo que não tem a licença totalmente livre.
- *nonfree*: utiliza algum sistema proprietário que complica a distribuição livre.

Sabendo dessa configuração, podemos utilizar o arquivo *sources.list* da forma que desejarmos, alterando livremente seus valores para sites que tenham menor latência ou que tenham outras fontes de softwares alternativos.

Além das áreas de licenças, é possível detalhar ainda mais a classe de um software. Para isso o padrão do APT sugere a organização de seções que determinam a finalidade do pacote. Muitas vezes as áreas e seções se fundem em um único parâmetro. Porém, quando existentes, estas seções podem variar para repositórios não oficiais. Sobretudo o projeto *Debian* publica as seções conforme a natureza do aplicativo como por exemplo: *audio, multimedia, office, development*, entre outros.

- /etc/apt/preferences

Esse arquivo contém informações sobre qual é o grau de softwares padrão para os comandos *apt-get*. Exemplo disso, é quando se deseja instalar qualquer programa sem informar o grau (*testing, unstable ou stable*) e então o *apt* pode solicitar qualquer o grau padrão definido no arquivo de preferencia. Um situação hipotética poderia conter o seguinte *preferences*:

*Package: **
Pin: release a=testing
Pin-Priority: 900
*Package: **
Pin: release a=unstable
Pin-Priority: 800

Este arquivo acima informaria que todos os pacotes *testing* tem preferencia maior em relação aos pacotes *unstable.* Desta forma, qualquer comando "*apt-get install*" faria a instalação através do repositório de *testing*. Nesse caso para forçar o uso de um pacote da versão *unstable* o comando *apt-get* deve ser usado em conjunto com o parâmetro " *-t unstable*".

/var/lib/apt/lists: O diretório de listas contém os pacotes encontrados em cada uma das *URIs* existentes no arquivo de fontes (*sources.list*). Toda vez que se realiza uma operação *update*, essas listas recebem os pacotes disponíveis nos repositórios, bem como as informações perinentes.

#> apt-get update

O update tem como objetivo atualizar as listas de softwares de softwares disponíveis fazendo a leitura do arquivo */etc/apt/sources.list*. É importante pois mantém a referência da listagem de softwares existentes nos repositórios configurados no sistema.

#> apt-get install <pacote>

Faz a instalação do pacote escolhido. Leva em consideração as entradas de preferências do *apt* já mencionadas anteriormente.

#> apt-get remove <pacote>

Remove um pacote já instalado.

#>apt-get dist-upgrade

Faz a atualização da versão inteira do software, migrando a distribuição inteira de uma só vez. Isto é, se a versão de software atual se refere ao *stable*, o comando *dist-upgrade* o levará para uma atualização ao *testing*.

#>apt-get upgrade

Dentro da distribuição atual, o comando upgrade fará a atualização de todo o software existente no sistema. Fará a instalação das últimas versões disponíveis para todo os pacotes.

#>apt-cache search <pacote>

Faz a busca dentro de suas referências por um determinado pacote.

12.2.4 – Criando seu próprio pacote deb

Para criar um pacote *debian* é bastante simples. Para tal, basta fazer uso do aplicativo *dpkg-deb*. O passo abaixo mostra um exemplo básico. A partir deste exemplo pode se entender a idéia para o processo de criação e a partir daí criar pactes maiores ou úteis.

O primeiro passo consiste em criar um diretório base:

```
#> mkdir /tmp/pacote
```

Depois é necessário criar um diretório que vai conter o arquivo de informações do pacote. Por padrão esse diretório leva o nome de *DEBIAN* e o arquivo leva o nome de *control.*

```
#> mkdir /tmp/pacote/DEBIAN
#> touch /tmp/pacote/DEBIAN/control
```

Em seguida, deve-se adicionar os campos exigidos para o arquivo de controle. São eles:

```
Package: pacote
Priority: optional
Version: 0.1
Architecture: i386
Maintainer: Rodrigo Zuolo Carvalho
Depends:
Description: Este é um pacote teste.
```

Onde os atributos em cada uma das linhas já intuitivamente já traduzem o significado próprio de cada um deles. Pois dependências, versão, mantenedor são todas informações referentes ao próprio pacote que se está criando.

Após a criação do arquivo de controle, é necessário popular o diretório base (/tmp/pacote) com os arquivos a serem gerados na instalação do pacote. Se em nosso exemplo, desejássemos criar um pacote que adicionaria um binário de nome *pacote* dentro de */usr/bin*, então seria necessário criar um diretório *usr/bin* dentro de /*tmp/pacote* e adicionar o arquivo binário pacote lá dentro, como no exemplo:

```
#> cd /tmp/pacote
#> mkidr -p usr/bin
#>cp ~zuolo/soft_do_pacote.bin /tmp/pacote/usr/bin/
```

Estes comandos colocariam o binário de nome *soft_do_pacote.bin* dentro /usr/bin em uma instalação deste pacote em questão.

Para finalizar a criação do pacote, é necessário então executar o comando *dpkg-deb*, conforme a linha a seguir.

```
#>dpkg-deb -b /tmp/pacote /tmp/
```

Com isso, o pacote cujo arquivo de instalação é *pacote.deb* será criado no diretório /*tmp*

Ao se instalar esse novo pacote, teremos um binário de nome *soft_do_pacote.bin* dentro de /*usr/bin*/. Isso pode ser feito com o comando de instalação padrão.

```
#> dpkg -i  /tmp/pacote.deb
```

12.2.5 – Criando seu próprio repositório APT

Além de criar um pacote, nos é possível exercitar a criação de um repositório próprio ou um espelho de pacotes *deb*. Isso pode ser útil quando se deseja criar um servidor para distribuir aplicações de uso local e específico ou quando se deseja replicar um servidor original para um acesso mais rápido à determinados clientes mais próximos.

Abaixo, o passo a passo para fazer a criação de um repositório contendo o nosso pacote criado no exemplo do item anterior.

Primeiramente deve-se criar um diretório de pacotes. Nesse caso, vamos usar o diretório root como base.

```
#> mkdir /root/debs
#> mkdir -p /root/debs/dists/squeeze/main/binary-i386/
```

Atenção deve ser dada ao diretório que segue o *dists.* Nesse caso leva o nome de *squeeze* pois estamos criando um repositório para a versão 5 do Debian.

Dentro desse diretório, acima criado, podemos inserir os pacotes binários a serem distribuídos. Um exemplo seria adicionar o pacote anteriormente criado.

```
#> cp /tmp/pacote.deb /root/debs/dists/squeeze/main/binary-i386/
```

Feito isso, é necessário criar um arquivo de *release* na base do diretório que contém os binários. O arquivo criado é o *Release*. O seu conteúdo pode ser visto nas linhas:

```
Arquive:squeeze
Component:main
Origin:Debian
Label:local
Architeture:i386
```

Com o arquivo de lançamento pronto, é necessário gerar o índice dos *pacoctes*. Para tal deve-se rodar o aplicativo *dpkg-scanpackages*, gerando uma lista de pacotes em um formato compactado para otimizar o uso de espaço. É importante salientar que o aplicativo *dpkg-scanpackages* está contido no pacote *dpkg-dev*. Portanto, antes de utilizá-lo, deve-se verificar se o mesmo se encontra instalado.

Existe o arquivo opcional de *override* que indica quais as seções de um software. Ele é útil para auxiliar as ferramentas amigáveis de instalação que segmentam os software por seções. O mesmo deve ficar situado no diretório base dos pacotes e a sua sintaxe é:

<pacote> <prioridade> <seção>

Um exemplo de linhas para o arquivo de *override* poderiam ser assim descritas.

```
adobe-flashplugin   low          web
quake               medium       games
```

A criação do arquivo *override* é opcional. Caso o mesmo tenha sido criado, então no momento da criação do arquivo de índices é necessário informar o arquivo de *override*. Para criar o arquivo de pacotes, será usada a ferramenta já citada, *scanpackages*.

```
#> dpkg-scanpackages /root/debs/dists/squeeze/main/binary-i386/  override > Packages
#> cd /root/debs/dists/squeeze/main/binary-i386/debs/
#> gzip -c Packages >  Packages.gz
```

Uma vez criado os índices para os pacotes em *packages.gz*, é necessário adicionar à sua fonte local de pacotes, descrita em */etc/apt/sources.list*.

Adicionando a linha:

```
deb file:/root/debs/ squeeze main
```

A partir deste ponto, a ferramenta *apt* pode ser atualizada para que a mesma possa procurar por pacotes também na base local, existente em */root/debs* .

12.3 - Bibliografia

- Diversos, The GNU Awk User's Guide. Acessado em 22 de setembro de 2009, http://www.gnu.org/software/gawk/manual/gawk.html

- Hamish Whittal. Shell Script (2005). Acessado em 22 de setembro de 2009, http://learnlinux.tsf.org.za/courses/build/shell-scripting/index.html